INFORMATION AND CONTROL 18, 483–501 (1971)

On Restricting the Base Component of Transformational Grammars*

P. STANLEY PETERS, JR.

University of Texas at Austin

AND

R. W. RITCHIE

University of Washington

We investigate the effects of placing various restrictions on the base component of a transformational grammar as defined by Chomsky (1965). It is shown that by utilizing the so-called filter function of transformations the descriptive power of transformational grammars can be preserved unreduced even when their base components are subjected to drastic restrictions.

1. INTRODUCTION

In Peters and Ritchie (1971) we defined in a mathematically precise fashion the notion *transformational grammar*, modeling essential features of these grammars as discussed informally in Chomsky (1965). Such a grammar was defined to contain a *base component* consisting of a phrase structure grammar —an unordered set of rewriting rules of the type $A \rightarrow \omega / \varphi - \psi$ (the symbol A may be rewritten as the (nonnull) string ω if it appears between φ, ψ). These rules allow one to derive strings of terminal symbols from the initial string $\#S\#$ and to assign one or more labeled bracketings to each derived string showing its composition as a sequence of phrases. The resulting set of labeled bracketings (*Phrase-markers*) serves as input to the *transformational component* of a transformational grammar, which consists of a linear sequence of *grammatical transformations* each of which converts labeled

* This work was supported in part by the Advanced Research Seminar in Mathematical Linguistics, 1968 and 1969, sponsored by the Center for Advanced Studies in the Behavioral Sciences, Stanford, California. The first author also received support from NSF Grant GS 2468.

bracketings into labeled bracketings in a manner we briefly summarize here. A grammatical transformation consists of a *structural condition* and a set of *elementary transformations*. An input labeled bracketing is first factored into a sequence of subparts so that the factorization satisfies the structural condition; such a factorization is called a *proper analysis* of the labeled bracketing for the transformation. If the input labeled bracketing has no proper analysis, the transformation gives the input labeled bracketing as output. Given a proper analysis of the input, the elementary transformations are applied to its factors. Each elementary transformation will either a) *delete* a sequence of factors, b) *substitute* one sequence of factors for another, or c) *adjoin* one sequence of factors to the right (left) of another. A structural condition and set of elementary transformations must meet a condition of compatibility called the *condition of recoverability of deletions* in order to be a transformation. Given a labeled bracketing produced by the base, transformations are applied cyclically to the subsentences of the labeled bracketing beginning with the most deeply embedded sentence, i.e., the subsentence which terminates with the leftmost $]_S$ in the labeled bracketing. In accordance with the principle of the *transformational cycle*, all transformations are applied in order to this subsentence and, after completion of each cycle, the transformations are applied next to the subsentence terminating with the next $]_S$ to the right. This process is iterated until the cycle has operated upon the outermost subsentence. The resulting labeled bracketing is a *surface structure* if it meets a condition of well-formedness, namely, that it contain no occurrences of the boundary symbol #. A labeled bracketing φ produced by the base is said to be a *deep structure* underlying the surface structure ψ if φ can be converted by the transformational component into ψ. The pair (φ, ψ) is a *structural description* generated by the grammar, and is assigned by the grammar to the sequence of terminal symbols which results from deleting the labeled brackets in ψ. The set of strings of terminal symbols to which the grammar assigns structural descriptions is called the *language generated* by the grammar, and the strings in the language are called *sentences*.

Notice that not every labeled bracketing produced by the base need be a deep structure. The base rules may introduce the boundary symbol # into a labeled bracketing from which it cannot be deleted in the course of a transformational derivation. The transformational component is said to have *filtered out* such a labeled bracketing from the set of labeled bracketings produced by the base component. Thus the deep structures are precisely those labeled bracketings produced by the base component which are not filtered out by the transformational component (see Fig. 1). Theorem 5.1 of Peters and Ritchie (1971) tells us that the class of transformational gram-

mars generates all the recursively enumerable languages. In the proof of this theorem, context-sensitive grammars whose rules did not introduce # were used as base components of the grammars constructed. Thus no use need be made of the filter function of transformations in order to generate all recursively enumerable languages if the base component is allowed to be

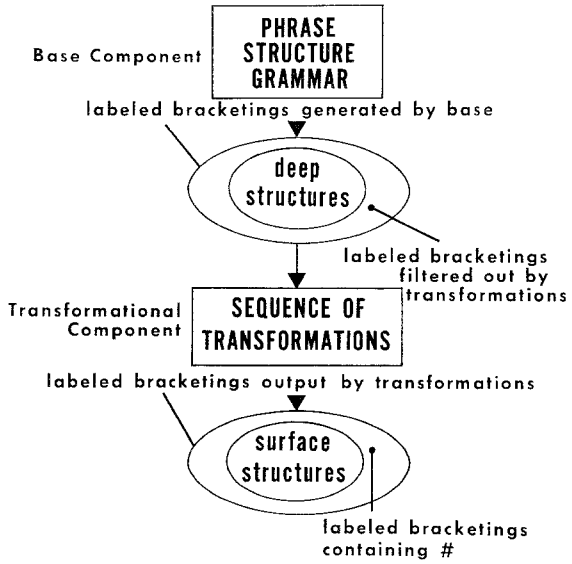

FIGURE 1

any context-sensitive grammar. In this paper, we investigate the effects of restricting the base component of transformational grammars, heavily exploiting the filter function of transformations. We do not consider here the interesting question whether the theory of transformational grammars would be essentially different if the filter function of transformations were eliminated (i.e., if base rules were not allowed to introduce the boundary symbol).

The major results of Peters and Ritchie (1971) concerned the sets of languages generated by certain classes of transformational grammars. In that paper, the base component was allowed to be an arbitrary context-sensitive grammar and restrictions were placed on the transformational component, after noting that the full class of transformational grammars generates exactly the recursively enumerable languages. The class of grammars having (total) recursive cycling functions (cf. Definition 6.5 of that paper) was shown to generate exactly the recursive languages. The stronger restriction that

a grammar have a cycling function belonging to a certain class (primitive recursive, elementary recursive, etc.) yields grammars which generate only languages with characteristic functions in the same class. We are now able to show, examining the converse question, that each member of these classes of languages is generated by a transformational grammar whose cycling function is in the same class (cf. Theorem 2).

As we turn to restricting the base component, several natural possibilities suggest themselves. We might ask, for example, what class of languages can be generated by transformational grammars with context free bases, with minimal linear bases or with one-sided linear bases. We might even ask whether a class of transformational grammars having a fixed base component can generate an interesting class of languages. A curiosity of our proofs of the results summarized above was that a single, fixed transformational component sufficed for the generation of all languages in the classes for which we obtained characterizations, each distinct language requiring a different base component in its grammar. While this result does not appear to have any linguistic significance, a parallel question, known as the Universal Base Hypothesis, is of considerable linguistic importance. Some linguists hypothesize that transformational grammars of all natural languages have the same base component. Our results of this paper bear on this hypothesis because they show that restricting the base component, even very stringently, does not restrict the class of languages generated by transformational grammars.

2. Preliminary Results

The fundamental observation from which our results follow is that a sequence of transformations can be used to carry out a computation by an arbitrary Turing machine. This fact is established in Lemma 1, for which it will be convenient to have in mind a specific model of Turing machines. A *Turing machine* Z over the alphabet $A = \{a_1, ..., a_n, b\}$ (b is the "*blank*" symbol) is taken as in Davis (1958, pp. 4–7) to be a finite set of quadruples each of which has the form (s_p, c, d, s_q) or (s_p, c, R, s_q) or (s_p, c, L, s_q), where there is a positive integer r (the number of states) such that $1 \leqslant p, q \leqslant r(\{s_1, ..., s_r\}$ is the set of states and s_1 is the *initial* state), $c, d \in A$ and for every pair (s_p, c), there is at most one quadruple beginning with this pair. The three forms of quadruple mean that when in state s_p scanning c the machine will enter state s_q and either print d, shift right one square, or shift left one square. The Turing machine Z started in state s_1 scanning the leftmost nonblank symbol of byb when $y \in \{a_1, ..., a_n\}^*$ goes through

a series of *steps* (one instantaneous description yielding another in the terminology of Davis (1958, p. 7)) and if this sequence reaches a configuration in which no quadruple of Z is applicable, a *computation* has been performed. The output tape is a sequence, possible the empty sequence, of symbols of $\{a_1, \ldots, a_n\}$ with b's interspersed. The sequence of a_i's which results from the deletion of the b's is denoted $Z(y)$, and we define the *language enumerated* by Z to be the set of all $x \in A^*$ for which there is a $y \in \{a_1, \ldots, a_n\}^*$ such that $x = Z(y)$.

Our desire is to shift as much as possible of the complexity required to generate arbitrary recursively enumerable languages to the transformational component of grammars so that the base component can be greatly restricted. In Lemma 1 we show that transformations can perform a step-by-step simulation of the computations of an arbitrary Turing machine and, since every recursively enumerable language is enumerated by a Turing machine, the base component need only set up the input tape and thus can be a linear grammar. Lemma 2 establishes that the transformational component can even set up the input tape, if the base component makes a copy of the alphabet available. Theorem 1 combines Lemmas 1 and 2 and shows that as a result the base component can be made extremely simple.

For purely technical purposes, it is convenient in the statements of both lemmas to refer to the same special property of labeled bracketings. Because of this property's repeated occurrence and rather lengthy description, we designate it as the "subsentence property" for purposes of reference in this paper. A labeled bracketing φ is said to possess the *subsentence property* if it has a leftmost subsentence (i.e., a subsentence whose $[_S$ is to the left of all terminal symbols in φ) which either ends with the sequence $\#\#\#\#b$ of terminals and contains exactly one occurrence of b to the left of this string or has as its only terminal symbol a single occurrence of $\#$.

LEMMA 1. *To every Turing machine Z there is a linear set $\mathscr{B}$ of labeled bracketings and a sequence $\mathscr{T}$ of transformations such that a string x is in the language enumerated by Z if and only if*

(1) *x does not contain the symbol $\#$ and*

(2) *x is the debracketization of the last line of some transformational derivation with respect to $\mathscr{T}$ having a member of $\mathscr{B}$ as its first line.*

In fact, $\mathscr{B}$ and $\mathscr{T}$ may be chosen so that (a) *$\mathscr{B}$ is the set of all labeled bracketings of the form*

$$[_S[_S \cdots [_S[_S[_S \underbrace{\# \cdots \#}_{r} a_1 \cdots a_n b\#\#\#\#b]_S \underbrace{b \cdots b}_{u}\#y\underbrace{b \cdots b}_{v}]_S\#]_S \cdots \#]_S\#]_S \tag{3}$$

where r is the number of states of Z, $\{a_1, \ldots, a_n, b\}$ is the alphabet of Z with b being the "blank" symbol, y is an arbitrary nonempty string over $\{a_1, \ldots, a_n\}$, and u and v are positive integers;[1] *and* (b) *every transformation of $\mathscr{T}$ effects the identity mapping on any labeled bracketing such that the subsentence being cycled on does not possess the subsentence property*; *and* (c) *there is a* 1—1 *association between computations of Z on the one hand and transformational derivations with respect to $\mathscr{T}$ beginning with a member of $\mathscr{B}$ and terminating with a #-free bracketing, on the other hand, such that a computation is associated with a derivation which has the output of the computation as the debracketization of its last line and which involves $k + m + 2$ cycles, where k is the number of steps in the associated computation and m is the length of tape used.*

Proof. We begin by showing that the first sentence of the lemma is a consequence of the second. Note that the set of all labeled bracketings of form (3) is a linear set, so that it remains to show that the existence of the 1—1 association asserted as (c) implies that properties (1) and (2) hold of a string x if and only if it is enumerated by Z. But if x is enumerated by Z, then there is a computation of Z with x as output and a transformational derivation associated with the computation as desired. Conversely, if x is #-free and is the debracketization of the last line of a derivation from a string in $\mathscr{B}$, then the associated computation by Z has x as output so that x is in the language enumerated by Z.

It suffices then to take $\mathscr{B}$ to be the set of all labeled bracketings of the form (3) and to construct a sequence $\mathscr{T}$ of transformations satisfying conditions (b) and (c).

The sequence $\mathscr{T}$ will be constructed so that the computation of Turing machine Z given input y will be mimicked step by step in the transformational derivation with (3) as first line. Simulation of Z on (3) is accomplished by the single transformation T_7.[2] The typical Phrase-marker during the course of the action of transformation T_7 will have the form (4),

$$[_S[_S \cdots [_S[_S[_S \underbrace{\# \cdots \#}_{r} a_1 \cdots a_n b \underbrace{\# \cdots \#}_{t} b]_S \, by_1 \underbrace{\# \cdots \#}_{p} cy_2 b]_S \#]_S \cdots \#]_S \#]_S \tag{4}$$

[1] The contents of the innermost subsentence of (3) are present for technical convenience, and the # to the left of y signals that Z is scanning the leftmost symbol of y in initial state s_1. Labeled bracketings of the form (3) will underlie a #-free Phrase-marker only if u and v are the numbers of tape square to the left and right respectively of the input y used by Z in its computation. These points will be clarified in the subsequent discussion of the transformational rules.

[2] The first transformation in $\mathscr{T}$ which we discuss is called T_7 because six transformations will eventually precede it; two will be discussed soon and four others will not be introduced until Lemma 2.

where y_1cy_2 is the tape of the Turing machine Z at the step being imitated and Z is in its p-th state, s_p, (among its states $s_1, \ldots, s_r$) scanning the symbol c indicated and where $t = 4, 5, 6$, or 7. In simulating an elementary step by Z, the transformation T_7 must replace $\underbrace{\# \cdots \#}_{p}$ in (4) by $\underbrace{\# \cdots \#}_{q}$ (changing the state from s_p to s_q) and perform the one of the three other operations allowed which is determined by the quadruple of Z in question. A hint of the structural condition and a specification of the elementary transformations of T_7 are

$$T_7 : [_S\underbrace{\# \cdots \#}_{q} - U_1 - U_2 - U_3 - (\#(\#(\#))) \; \#\#\#\#b]_S$$

$$\begin{array}{cccccccccccccccc} & & & & & 1 & & 2 & 3 & 4 & & & 5 & & & \\ -X- & U_4 & -e- & U_5 & - & \underbrace{\# \cdots \#}_{p} & - & e & - & U_6 & - & U_7 & - & U_8 & - & Y & - & b & - & \# \\ 6 & 7 & 8 & 9 & & 10 & & 11 & 12 & & 13 & & 14 & & 15 & & 16 & & 17 \end{array}$$

substitute 13 for 8
substitute 1 for 10
substitute 9 for 11
substitute 3 for 14
delete 9
delete 13
delete 17

The values of the "dummy variables" U_i are determined by the quadruples of Z. We will employ the following convention in describing the structural condition of T_7: the predicate $i \to i \equiv^{17} x$ (cf. Peters and Ritchie, 1971, Definition 2.12) will be abbreviated as $i = x$. The structural condition is the conjuction of the six sentences "$S^{17}_{1\to 5}$", "$5 = \#\#\#\#b$ or $5 = \#\#\#\#\#b$ or $5 = \#\#\#\#\#\#b$ or $5 = \#\#\#\#\#\#\#b$", "$8 = e$", "$11 = e$", "$16 = b$", and "$17 = \#$" with a long disjunction containing one disjunct for each quadruple in Z. We now describe how to obtain the disjuncts.

Case I. If the quadruple is a print instruction (s_p, c, d, s_q), then setting $a_{n+1} = b$ and i such that $d = a_i$ the disjunct is "$1 = \underbrace{\# \cdots \#}_{q}$ and $10 = \underbrace{\# \cdots \#}_{p}$ and $2 = \underbrace{\# \cdots \#}_{r-q} a_1 \cdots a_{i-1}$ and $3 = a_i$ and $4 = a_{i+1} \cdots a_{n+1}$ and $14 = c$ and $13 = e$ and $9 = e$ and $12 = e$ and $(7 = a_1$ or $\cdots$ or $7 = a_n$ or $7 = b)$"; i.e., $U_2 = d$ (the symbol to be printed), $U_8 = c$ (the symbol to be overprinted) and $U_7 = e$, $U_5 = U_6 = e$ and U_4 is one of $a_1, \ldots, a_n, b$.

Case II. If the quadruple is a move right instruction (s_p, c, R, s_q), then the disjunct is "$1 = \underbrace{\# \cdots \#}_{q}$ and $10 = \underbrace{\# \cdots \#}_{p}$ and $13 = c$ and $2 = \underbrace{\# \cdots \#}_{r-q} a_1 \cdots a_n\, b$ and $3 = e$ and $4 = e$ and $14 = e$ and $9 = e$ and $12 = e$ and $(7 = a_1$ or $\cdots$ or $7 = a_n$ or $7 = b)$"; i.e., $U_7 = c$ (the symbol to be moved right off of) and $U_2 = U_8 = e$, $U_5 = U_6 = e$ and U_4 is one of $a_1, \ldots, a_n, b$.

Case III. If the quadruple is a move left instruction (s_p, c, L, s_q), then the disjunct is "$1 = \underbrace{\# \cdots \#}_{q}$ and $10 = \underbrace{\# \cdots \#}_{p}$ and $12 = c$ and $(9 = a_1$ or $\cdots$ or $9 = a_n$ or $9 = b)$ and $2 = \underbrace{\# \cdots \#}_{r-q} a_1 \cdots a_n\, b$ and $3 = e$ and $4 = e$ and $14 = e$ and $13 = e$ and $7 = e$"; i.e., $U_6 = c$ (the symbol to be moved left off of) and U_5 is one of $a_1, \ldots, a_n, b$ and $U_2 = U_8 = e$, $U_7 = e$ and $U_4 = e$.

In deciphering T_7, it may be helpful to notice that U_6, U_7 and U_8 are respectively the scanned square in simulating moves of the types "left", "right" and "print" and that U_4 is present to assure both that the sequence of p #'s in term 11 is not a proper substring of the string representing the state of Z (hence the state of Z really is s_p) and also that no move is made if the leftmost b of the tape is under scan. Term 16 simply guarantees that no move is made if the rightmost b of the tape is being scanned, term 17 with its associated deletion elementary transformation accomplishes the erasure of the new boundary symbol which appears as each cycle is taken in the transformational derivation and terms 1 and 10 accomplish the state change. A print instruction is accomplished by the replacement of term 14 (U_8) by term 3 (U_2), a move right instruction by the replacement of term 8 (e) by 13 (U_7) and a move left instruction by the replacement of term 11 (e) by term 9 (U_5).

It is straightforward to check that T_7 imitates Z when applied to Phrase-markers such as (4), except in the moves labeled (3) and (5) by Davis (1958, p. 7), which add new tape squares if the edge of the tape is reached. If the end of the input tape as set up in (3) is reached, the structural condition of T_7 is not satisfied and the reader can easily check that it and each of the transformations T_5, T_6, T_8, T_9 and T_{10} defined below apply vacuously on all remaining cycles so that the Phrase-marker produced contains #'s. We now introduce T_5 and T_6 which utilize the value of t in labeled bracketings of the form (4) to record whether or not Z's computation uses all the tape provided except the leftmost and rightmost squares. The value of t, initially 4, is increased by 1 when the leftmost usable square of tape is reached for the first time (T_5) and by 2 when the rightmost usable square is reached for the

first time (T_6), so that t equals 7 just in case both ends but for one square have been reached.

$$T_5 : [_S \underbrace{\# \cdots \#}_{r} a_1 \cdots a_n b \# \# \# - \# - (\# \#) b]_S - b \# - Y - \#$$

$$\begin{array}{cccccc} 1 & & 2 & 3 & 4 & 5 & 6 \\ 1+2 & & 2 & 3 & 4 & 5 & 6 \end{array}$$

$$T_6 : [_S \underbrace{\# \cdots \#}_{r} a_1 \cdots a_n b \# \# - \# \# - (\#) b]_S - Y - \# \begin{Bmatrix} a_1 \\ \vdots \\ a_n \\ b \end{Bmatrix} b \#$$

$$\begin{array}{cccccc} 1 & & 2 & 3 & 4 & 5 \\ 1+2 & & 2 & 3 & 4 & 5 \end{array}$$

When a configuration of form (4) is reached representing an instantaneous description of Z in which Z halts and in which all the tape provided has been used, then the clean-up transformations T_8, T_9, and T_{10} become applicable. Transformation T_8, which will not apply on any cycle on which T_7 has applied (nonvacuously) since T_8 requires # as rightmost symbol and T_7 deletes the rightmost #, checks that the squares at each end of the tape are not being scanned—guaranteeing that the inapplicability of T_7 was not occasioned by Z's running off the edge of the tape—and also checks that that $t = 7$, insuring that no excess tape was provided. If these conditions are met, then T_8 erases the b in the right end square and the sequence of #'s representing the state in which Z halted, replaces the deepest subsentence by #, and positions a single # on the tape to signal applicability of T_9. T_9 passes a # across the tape from left to right, one square at a time erasing each b encountered. After T_9 has passed the # across all of the tape but the rightmost symbol, T_{10} erases this # (and the rightmost symbol if it is b), leaving exactly the string on $a_1, \ldots, a_n$ output by Z.

$$T_8 : [_S \underbrace{\# \cdots \#}_{r} a_1 \cdots a_n b \# \# \# \# \# \# - \# - b]_S - X$$

$$\begin{array}{cccccc} 1 & & & 2 & 3 & 4 \\ 0 & & & 2 & 0 & 2+4 \end{array}$$

$$- \begin{Bmatrix} a_1 \\ \vdots \\ a_n \\ b \end{Bmatrix} - \underbrace{\# \cdots \#}_{1 \leqslant j \leqslant r} - \begin{Bmatrix} a_1 \\ \vdots \\ a_n \\ b \end{Bmatrix} - Y - b \#$$

$$\begin{array}{ccccccc} 5 & & 6 & & 7 & 8 & 9 \\ 5 & & 0 & & 7 & 8 & 0 \end{array}$$

$$T_9 : [_S\#]_S - X - \# - \left\{ \begin{matrix} b \\ e - \left\{ \begin{matrix} e \\ a_1 \\ \vdots \\ a_n \end{matrix} \right\} \end{matrix} \right\} - Y - \#$$

1	2	3	4	5	6	7
1	2	0	0	$5+3$	6	0

Condition: $6 \neq e$.

$$T_{10} : [_S\#]_S - X - \# - \left\{ \begin{matrix} b \\ e - \left\{ \begin{matrix} e \\ a_1 \\ \vdots \\ a_n \end{matrix} \right\} \end{matrix} \right\} - \#$$

1	2	3	4	5	6
0	2	0	0	5	0

Returning to the statement of Lemma 1, we recall that we have taken $\mathscr{B}$ to be the set of labeled bracketings of the form (3) as promised in (a). We now take $\mathscr{T}$ to be the sequence $(T_5, T_6, T_7, T_8, T_9, T_{10})$ of transformations just constructed, note that (b) is satisfied and complete the proof by showing that (c) also holds.

Note first that to every labeled bracketing φ in $\mathscr{B}$ there is a *unique* transformational derivation with respect to $\mathscr{T}$ having φ as its first line. We now associate with each computation by Z the unique derivation with respect to $\mathscr{T}$ beginning with the labeled bracketing φ of the form (3) in which

(i) y is the input beginning the computation by Z,

(ii) u and v are respectively the number of squares to the left and right of the input y which are used in this computation, and

(iii) there are exactly $k + u + v + l(y) + 2$ subsentences of φ, where k is the number of steps in the computation and where $l(y)$ is the length of y.

Note that, by the construction of $\mathscr{T}$, this derivation will have a #-free last line, since T_8 will apply nonvacuously on the $k+$ 3rd cycle and exactly $m - 1$ cycles later T_{10} will apply deleting the final #'s, where $m = u + v + l(y)$; further the debracketization of this last line is just the output of the computation. We must now show the converse to establish the 1—1 association asserted to exist in part (c) of the lemma; namely that to each derivation which begins on a φ of the form (3) and which has its last line #-free there is an associated computation by Z such that (i), (ii), and (iii) hold. To show this, observe that the last line of the derivation begun on any φ of the form (3) is #-free if and only if T_{10} applied nonvacuously on the final cycle; for each member

of $\mathscr{B}$ contains a # and the existence of the #'s is preserved by $T_5, \ldots, T_9$. But this happens if and only if T_8 applied nonvacuously $m - 1$ cycles earlier. Further, T_8 applies nonvacuously if and only if there is a preceding sequence of some number, k, of cycles, beginning with the 3rd cycle of the derivation, in which T_7 imitated step by step the unique sequence of steps by Z begun on input y and this sequence is an imitation of a computation by Z satisfying (i) and (ii). The number of subsentences is exactly $k + m + 2$ as desired, since after two initial vacuous cycles there are k cycles imitating the computation preceding the nonvacuous application of T_8 and $m - 1$ cycles following it, hence part (c) of the lemma is proved and the lemma has been established.

The necessity of generating the labeled bracketings of $\mathscr{B}$ does not require much complexity of the base component. But the role of this component can be reduced even further since the transformations can be made to set up the "input tape", as well as carry out a "computation" on it. We demonstrate this in Lemma 2.

LEMMA 2. *There is a minimal linear set $\mathscr{B}'$ of labeled bracketings and a sequence $\mathscr{T}'$ of transformations such that every labeled bracketing of the form* (3) *is the last line of a transformational derivation whose first line is in $\mathscr{B}'$ and further, every bracketing which is the last line of such a derivation but is not of the form* (3) *contains at least one # and also fails to possess the subsentence property. In fact, $\mathscr{B}'$ and $\mathscr{T}'$ may be chosen so that* (a) *$\mathscr{B}'$ is the set of all labeled bracketings of the form* (5);

$$[_S[_S \cdots [_S[_S\, a_1 a_2 \cdots a_n b\#]_S\#]_S \cdots \#]_S\#]_S \tag{5}$$

(b) *every transformation in $\mathscr{T}'$ effects the identity mapping on each labeled bracketing which possesses the subsentence property; and* (c) *in every transformational derivation from a labeled bracketing of type* (5) *to a labeled bracketing of the form* (3), *at least one transformation in $\mathscr{T}'$ applies nonvacuously on each of the first $m + 1$ cycles, where $m = u + v + l(y)$.*

Proof. We take $\mathscr{B}'$ to be the set of strings of form (5) and construct the four transformations which constitute $\mathscr{T}'$. Transformation T_1 produces the r boundaries for the innermost subsentence and T_2 positions them correctly, and adds a b as the rightmost symbol. Both apply nonvacuously only on the first cycle of a derivation.

$$\begin{array}{lccccccccc} T_1: & a_1 & \cdots & a_n & - & e & - \cdots - & e & - & b & - & \# \\ & 1 & & & & 2 & & r+1 & & r+2 & & r+3 \\ & 1 & & r+3 & & r+3 & & r+2 & & r+3 & & \end{array}$$

$$T_2 : a_1 \cdots a_n - \underbrace{\# \cdots \#}_{r} - b - \#$$

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 2+1 & 0 & 3 & 4+3 \end{array}$$

Transformations T_3 and T_4 produce the other three #'s in the innermost subsentence and the string $b^u \# y b^v$.

$$T_3 : [_S \# - \underbrace{\# \cdots \#}_{r-1} a_1 \cdots a_n - b - \begin{Bmatrix} e \\ \# \end{Bmatrix} - \begin{matrix} \# \\ e \end{matrix} - (\#\#)\, b]_S - X - \#$$

$$\begin{array}{ccccccccc} 1 & & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & & 2 & 3 & 4 & 4+5 & 6 & 3+7 & 0 \end{array}$$

Condition: 1 is not an S.

$$T_4 : [_S \# - \underbrace{\# \cdots \#}_{r-1} a_1 \cdots a_{i-1} - \underset{1 \leqslant i \leqslant n}{a_i} - a_{i+1} \cdots a_n b$$

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{array}$$

$$- \begin{Bmatrix} e \\ \# \end{Bmatrix} - \begin{matrix} \# \\ e \end{matrix} \#b]_S - e - X - \#$$

$$\begin{array}{ccccccc} 5 & 6 & 7 & 8 & 9 \\ 5 & 5+6 & 5 & 3+8 & 0 \end{array}$$

Condition: 1 is not an S.

On the second cycle and an arbitrary number of following cycles, T_3 will apply producing the string b^v in v cycles. On all but the last of these cycles, the fourth factor of proper analyses for T_3 will be the empty string. As soon as the fourth factor is taken as #, a second boundary is added in the innermost subsentence and T_3 becomes temporarily inapplicable. Transformation T_4 is applicable as long as the innermost subsentence contains two #'s at its right and applies on the v + 2nd and following cycles until term 5 of its proper analysis is taken as #. At this point a third # is added to the innermost subsentence, the string $\#y$ has been positioned to the left of b^v and T_3 once again becomes applicable. Transformation T_3 can produce b^u and add the fourth # in the innermost subsentence in u more cycles. Thus if φ is any labeled bracketing of the type (3) then, letting s be the number of subsentences of φ, there is clearly a transformational derivation begun on the labeled bracketing of the type (5) with exactly $m + s - 1$ subsentences, such that φ is the last line of the derivation, and nonvacuous applications of transformations occur on exactly the first $m + 1$ cycles, establishing property (c). That $\mathscr{T}'$ satisfies (b) is immediate by inspection of $T_1, \ldots, T_4$. Finally,

notice that every line in every derivation contains at least one # and fails to satisfy the subsentence property until an application of T_3 is made to a labeled bracketing in which the leftmost subsentence ends in $\#\#\#b$ and in which term 4 of its proper analysis is #. Since this results in a string of form (3), we see that if the last line of a derivation possesses the subsentence property, then it is also a line of form (3) as desired, completing the proof.

3. Restricted Bases

We now have the tools to attack the questions we have raised regarding restrictions on the base component. Our first result exhibits a highly restricted base component which is universal in the sense that every language that can be generated by a transformational grammar is generated by a grammar with this particular, fixed base component. This theorem also establishes the result announced as Theorem 5.2 in Peters and Ritchie (1971).

Theorem 1. *The following five conditions are equivalent for alphabet* $\{a_1, \ldots, a_n\}$:

(i) *L is a recursively enumerable language,*
(ii) *L is generated by a context-free based transformational grammar,*
(iii) *L is generated by a minimal linear based transformational grammar,*
(iv) *L is generated by a one-sided linear based transformational grammar,*
(v) *L is generated by a transformational grammar with the base component having rules $S \to S\#$, $S \to a_1 \cdots a_n b\#$, where b and # are terminal symbols not in $\{a_1, \ldots, a_n\}$, # being the boundary symbol.*

Proof. It is clear that (v) implies each of (iii) and (iv) and that they each imply (ii). That (ii) implies (i) follows *a fortiori* from Theorem 5.1 of Peters and Ritchie (1971). It remains only to show that (i) implies (v) to complete the proof, so let us assume that L is an r.e. language and construct a transformational grammar generating L with base component as in (v). Since L is r.e., there is a Turing machine Z enumerating L. Let $\mathscr{B}$ be the set of all labeled bracketings of the form (3) and let $\mathscr{T}$ be the sequence of transformations guaranteed by Lemma 1 to derive L from $\mathscr{B}$. Further, let $\mathscr{T}'$ be the sequence of transformations given by Lemma 2 which produces all the labeled bracketings of the form (3) from those of the form (5), i.e. from those strongly generated by the base component $S \to S\#$, $S \to a_1 \cdots a_n b\#$. We shall show that the transformational grammar with this base and with transformational component $\mathscr{T}''$ consisting of the transformations of $\mathscr{T}'$

followed by those of $\mathscr{T}$ generates exactly L. Let x be any element of L and let Z generate x from an input string y with a computation involving k steps, u squares to the left of y and v squares to the right. The transformational grammar generates x as follows. The base component produces the bracketing of the form (5) with exactly $k + 2m + 1$ subsentences where m is the sum of u, v and the length of y. By Lemma 2, the sequence $\mathscr{T}'$ of transformations can produce from this the labeled bracketing of the form (3) with $k + m$ subsentences surrounding

$$[_S[_S \underbrace{\# \cdots \#}_{r} a_1 \cdots a_n b \# \# \# \# b]_S \underbrace{b \cdots b}_{u} \# y \underbrace{b \cdots b}_{v}]_S .$$

But then so can $\mathscr{T}''$ since at each of the first $m + 1$ cycles in this derivation, the subsentence property cannot be possessed by the labeled bracketing acted upon (or else the derivation by $\mathscr{T}'$ would not continue, by Lemma 2 (b)) so that each transformation in $\mathscr{T}$ effects the identity mapping, by Lemma 1 (b). But then $\mathscr{T}$ applied to this labeled bracketing yields a #-free string whose debracketization is x, by Lemma 1 (c), and $\mathscr{T}''$ acts in exactly the same fashion since now the subsentence property is possessed by each line (by Lemma 1 (b)) so that each transformation in $\mathscr{T}'$ acts vacuously at each of these steps, by Lemma 2 (b). Hence every string in L is generated by the transformational grammar. For the converse, consider an arbitrary derivation by $\mathscr{T}''$ from any labeled bracketing of the form (5). Since any string in (5) fails to possess the subsentence property, the derivation is identical to one by $\mathscr{T}'$ until either a labeled bracketing is produced which possesses the subsentence property or the derivation terminates. By Lemma 2 the labeled bracketing φ produced at this point is either of form (3) or else fails to possess the subsentence property. If φ has form (3), then the derivation continues identically with the derivation from φ by $\mathscr{T}$ and thus concludes with a #-free labeled bracketing only if its debracketization is an element of L by Lemma 1. On the other hand, if φ is not of form (3), then the derivation results in a string which contains a #, hence is not a surface structure. Thus the only sentences generated are in L.

Thus if transformational components of grammars are allowed to vary freely, then restricting base components to contain only context-free rules, only left-linear rules or only right-linear[3] rules does not diminish the set of languages that can be generated. Transformations are such powerful devices

[3] Clearly the reflections of all recursively enumerable languages can be generated by grammars with a fixed right-linear base component and appropriate transformational components (simply reverse all base and transformational rules discussed in the proof of Theorem 1). But this is sufficient, since the set of recursively enumerable languages is closed under the operation of reflection.

that they can map highly restricted sets generated by very impoverished base components into arbitrary recursively enumerable languages. In fact, transformations are so powerful they can generate any recursively enumerable language from certain fixed sets of Phrase-markers, the outputs of certain particular grammars. Such universal base components are highly nonunique; some are trivial—as in Theorem 1 (v)—and some are very complex.

One infinite class of universal base components contains all linear grammars satisfying three conditions: a) every Phrase-marker generated contains at least one occurrence of the boundary symbol #, b) for every positive integer n a Phrase-marker is generated containing at least n subsentences and c) all symbols of the alphabet over which recursively enumerable languages are to be generated appear in each member of a set of Phrase-markers satisfying b) in such a way that the same structural condition can be used to identify them in each subsentence of every member of this set. The grammar of Theorem 1 (v) is a member of this class. While none of these grammars is linguistically natural, there are infinitely many universal base components which are not in this class. Grammars proposed by Anderson, Ross and Lakoff as universal base components suffice for the generation of every recursively enumerable language. It is likely, therefore, that every natural language can be generated from each of them (cf. Peters and Ritchie, 1971, Section 7). This fact points up the difficulty in empirically supporting or disconfirming the claim that all natural languages have the same base structure.

Turning now to a consideration of the transformational component one should note that the number of transformations in the grammar of the proof of Theorem 1 is of no particular significance. With minor alterations the ten transformations used in the proof can be collapsed to one. Clarity of presentation was the sole motive for using more. It is worth remarking that even with a fixed base component all recursively enumerable languages can be generated with very little variability in the transformational component of grammars. Only T_7 above depends in any essential way on the particular language to be generated. By changing the set-up part of the grammar, we can have T_7 simulate a fixed universal Turing machine and all the variability in the grammar can be confined to specifying the exact value of one constant term in the structural condition of one transformation. Thus the base and the transformational components can simultaneously be made universal in this way.

From the standpoint of empirical adequacy, a grammar must do more than simply generate the right strings as sentences. It must also, for example, assign the correct number of structural descriptions (degree of ambiguity) to each sentence. Transformational grammars have a remarkable flexibility in this regard also.

Remark. For any alphabet A, there is a one-sided linear grammar B with the following property. Given any recursively enumerable language L over A and any recursive function f into $\{1, 2, 3,...\} \cup \{\aleph_0\}$ with $L \subseteq \text{Domain } (f)$[4], there is a transformational grammar which has base component B, which generates L and which assigns every string x in L exactly $f(x)$ structural descriptions.

Since L is recursively enumerable, it is the range of some 1—1 recursive function g from the natural numbers into A^*. Thus there is a recursive function h the value of which is (a) $g(n)$ on $2^m \cdot 3^n$ if $1 \leqslant m \leqslant f(g(n))$ and (b) undefined on every other argument. The function h has the same range as g and maps onto a string x exactly $f(x)$ inputs. Taking Z as a Turing machine which computes h, we obtain the desired grammar by Lemma 1, since for each labeled bracketing in $\mathscr{B}$ there is exactly one transformational derivation with respect to $\mathscr{T}$. An immediate consequence is that every recursively enumerable language has an unambiguous transformational grammar. Clearly, the other empirical constraints on grammars deserve similar study.

4. Transformational Grammars with Bounded Cycling

Let us conclude with some results about the effect of restricting the cycling function $f_{\mathscr{G}}$ of a transformational grammar $\mathscr{G}$. Recall that if x is in $L(\mathscr{G})$, then $f_{\mathscr{G}}(x)$ is the smallest number s such that some deep Phrase-marker underlying x has exactly s subsentences, and $f_{\mathscr{G}}(x) = 0$ if x is not in $L(\mathscr{G})$. In Corollary 6.7 of Peters and Ritchie (1971) we showed that if $f_{\mathscr{G}}$ was bounded by a function which is elementary recursive (or primitive recursive, or in any one of Grzegorczyk's classes $\mathscr{E}^n$), then $L(\mathscr{G})$ is an elementary recursive language (primitive recursive language, language in $\mathscr{E}^n$), but we left open the converse. In Corollary 6.6 we showed that $f_{\mathscr{G}}$ was bounded by a recursive function if and only if $L(\mathscr{G})$ was recursive, but noted that the proof did not extend to subrecursive classes. Since the class of elementary recursive functions was shown to be equivalent to the class of "predictably computable" functions in Ritchie (1963), the converse question has a natural interpretation; namely, it excludes the existence of languages in which decisions of grammaticality can be made predictably, but in which these decisions cannot be reached by recreating the deep structure, which has unpredictably deep nesting of subsentences. We now prove two theorems

[4] One may interpret the strings of A^* as p-adic representations of natural numbers, where p is the cardinality of A. To avoid any aura of mysticism here about f taking $\aleph_0$ as a value, note that we may replace $\aleph_0$ in the remark by 0 if we modify the natural ordering $\leqslant$ just so that $n \leqslant 0$ for any natural number n.

which show that for every elementary recursive language, there is a grammar with elementary recursive cycling function (hence predictable deep structures), and also another grammar in which the cycling function is arbitrarily large (and hence the deep structure has unpredictably deep nesting).

THEOREM 2. *The following three conditions are equivalent for any language* L:

(i) L *is elementary recursive,*

(ii) *there exists a transformational grammar* $\mathcal{G}$ *such that* $L = L(\mathcal{G})$ *and* $f_{\mathcal{G}}$ *is elementary recursive,*

(iii) *there exists a transformational grammar* $\mathcal{G}$ *such that* $L = L(\mathcal{G})$ *and* $f_{\mathcal{G}}$ *is pointwise bounded by an elementary recursive function.*

The same holds if elementary recursive is replaced by primitive recursive or $\mathcal{E}^n$ *for any* $n > 3$, *where* $\mathcal{E}^n$ *is defined in Grzegorczyk* (1953).

Proof. We shall show the theorem for the elementary recursive case; the other cases being entirely similar. That (ii) implies (iii) is trivial since any elementary recursive function bounds itself. That (iii) implies (i) is proved in Corollary 6.7 of Peters and Ritchie (1971). Thus we need only show that (i) implies (ii), and to do so let us now assume that L is an elementary recursive set, i.e. that the characteristic function χ_L of L is elementary recursive, where $\chi_L(x) = 1$ if $x \in L$ and $\chi_L(x) = 0$ otherwise. If L is empty, the result is trivial, so assume that L is nonempty and that $x_0 \in L$. Let f be the elementary function that enumerates L as follows: on input x, $f(x) = x_0$ if $x \notin L$, while $f(x) = x$ if $x \in L$. (To see that f is elementary recursive, one might define f formally as $f(x) = x \cdot \chi_L(x) + x_0 \cdot (1 \dot{-} \chi_L(x))$ and appeal to the known closure properties of the elementary recursive functions as established in Grzegorczyk (1953).) Let Z be a Turing machine which computes f predictably, and let $\mathcal{G}$ be the transformational grammar for Z given by Lemma 1. Since Z computes predictably, the m and k given in part (c) of the lemma are elementary recursive functions of x (see Ritchie (1963) and Cobham (1965)). Thus $f_{\mathcal{G}}(x)$, which is in this case just $k + m + 2$, is elementary recursive, completing the proof.

This theorem contrasts with Corollary 6.7 of Peters and Ritchie (1971), which showed that in the case of the class of all recursive functions, the three conditions,

(i) $L(\mathcal{G})$ is recursive,

(ii) $f_{\mathcal{G}}$ is recursive, and

(iii) $f_{\mathcal{G}}$ is pointwise bounded by a recursive function,

were equivalent for *each* transformational *grammar*. Theorem 2 only asserts for the *language* L that elementary recursiveness is equivalent to the existence of at least one grammar $\mathcal{G}$ for L with appropriate $f_{\mathcal{G}}$. Theorem 3 shows that it is impossible to strengthen this result so that it becomes a statement about all grammars generating an elementary recursive set.

THEOREM 3. *To every elementary recursive set L, there is a transformational grammar $\mathcal{G}$ such that $L = L(\mathcal{G})$ and $f_{\mathcal{G}}$ is not bounded by any elementary recursive function. In fact, to every* r.e. *set L (including very simple* r.e. *sets, for example regular sets), and to every recursive function $f(x)$ (no matter how complex and rapidly growing, for example Ackermann's function (Ritchie, 1963, p. 272)), there is a transformational grammar $\mathcal{G}$ such that $L = L(\mathcal{G})$ and $f_{\mathcal{G}}(x) \geqslant f(x)$ for all x in L.*

Proof. The first assertion of this theorem follows from the second since, for example, any function g such that $g(x) \geqslant f(x)$ for infinitely many x, where $f(x)$ is Ackermann's function, is not primitive recursive (Ritchie, 1963, p. 272), hence certainly not elementary. To prove the second assertion, let L and f be given, and let Z be the Turing machine which enumerates L as follows. Let h be a recursive function having as range the r.e. set L. On input y compute $h(y)$ and $p^{f(h(y))+1}$, where p is the cardinality of Z's alphabet. Then erase the number $p^{f(h(y))+1}$ preserving the output $h(y)$ and halt. Lemma 1 produces a transformational grammar generating L for which the value of the cycling function is greater at each element $x = h(y)$ of L than the number of steps taken by Z in the computation described above. But this involves more than $f(x)$ steps (since $p^{f(x)+1}$ occupies at least $f(x) + 1$ squares of tape and hence its erasure requires more than $f(x)$ steps), so $f_{\mathcal{G}}(x) \geqslant f(x)$ as desired.

ACKNOWLEDGMENT

The authors acknowledge their indebtedness to Noam Chomsky, who introduced them both to the mathematical study of transformational grammars.

RECEIVED: April 15, 1970

REFERENCES

CHOMSKY, NOAM (1965), "Aspects of the Theory of Syntax," M.I.T. Press, Cambridge, Mass.

COBHAM, ALAN (1965), The Intrinsic Computational Difficulty of Functions, *in* "Logic, Methodology and Philosophy of Science," Proc. 1964 International Congress, North-Holland, Amsterdam.

DAVIS, MARTIN (1958), "Computability and Unsolvability," McGraw-Hill, New York.

GRZEGORCZYK, ANDREJ (1953), Some classes of recursive functions, *Rozprawy Matematyczne*. Warsaw.

KLEENE, S. C. (1952), "Introduction to Metamathematics," Van Nostrand, Princeton, N. J.

PETERS, P. STANLEY, JR. AND RITCHIE, R. W. (1971), On the generative power of transformational grammars, *Information Sciences*, to appear.

RITCHIE, ROBERT W. (1963), Classes of predictably computable functions, *Trans. Amer. Math. Soc.* **106**, 139–173.